DOCUMENT RESUME

ED 061 977 52 LI 003 647

AUTHOR Silver, Steven S.
TITLE INTX: Interactive Assembler Language Interpreter Users' Manual; Preliminary Programming Manual and Version II Extensions. Final Report.
INSTITUTION California Univ., Berkeley. Inst. of Library Research.
SPONS AGENCY Office of Education (DHEW), Washington, D.C. Bureau of Research.
BUREAU NO BR-7-1083
PUB DATE Sep 71
GRANT OEG-1-7-071083-5068
NOTE 34p.;(8 References)

EDRS PRICE MF-$0.65 HC-$3.29
DESCRIPTORS *Computers; *Electronic Data Processing; *Information Processing; Manuals; *On Line Systems; *Programing Languages
IDENTIFIERS Berkeley; *University of California

ABSTRACT

INTX is an interactive programing and debugging system operating under UCLA's URSA interactive console system. Although originally designed as a debugging aid for interactive processor development, the addition of an on-line Assembler makes it a programing system in its own right. INTX operates only on the Computer Communications 301 graphics display device making use of automatic update and cursor positioning facilities. There are three major divisions in the system: (1) Command analysis and initialization, (2) Assembler-loader and (3) Interpreter - disassembler. The Command module displays an initial screen describing functions available, current level of development, and supervises the operation of all other components of the system. The Assembler is a subset of Basic Assembler Language with extended branch mnemonics which accepts standard assembly language and the EQU pseudo operation. The Interpretor module is designed to execute any System/360 instruction except the SVC. Any instruction that could be successfully executed by a program running in problem state is supported. [Related documents are LI 003610, LI 003611, LI 003645, LI 003646 and LI 003648.] (Author/SJ)

FINAL REPORT
Project No. 7-1083
Grant No. OEG-1-7-071083-5068

INTX:
INTERACTIVE ASSEMBLER LANGUAGE INTERPRETER USERS' MANUAL

Preliminary Programming Manual*
and
Version II Extensions**

By

Steven S. Silver

Institute of Library Research
University of California
Berkeley, California 94720

September 1971

The research reported herein was performed pursuant to a grant with the Office of Education, U.S. Department of Health, Education, and Welfare. Contractors undertaking such projects under Government sponsorship are encouraged to express freely their professional judgment in the conduct of the project. Points of view or opinions stated do not, therefore, necessarily represent official Office of Education position or policy.

U.S. DEPARTMENT OF
HEALTH, EDUCATION, AND WELFARE

Office of Education
Bureau of Research

*date internal report produced: July 9, 1969
**date internal report produced: November 20, 1969

TABLE OF CONTENTS

PRELIMINARY PROGRAMMING MANUAL

VERSION II EXTENSIONS

Executer

Assembler

FOREWORD

This report contains the results of the second phase (July, 1968 - June, 1970) of the File Organization Project, directed toward the development of a facility in which the many issues relating to the organization and search of bibliographic records in on-line computer environments could be studied. This work was supported by a grant (OEG-1-7-071083-5068) from the Bureau of Research of the Office of Education, U.S. Department of Health, Education, and Welfare and also by the University of California. The principal investigator was M.E. Maron, Professor of Librarianship and Associate Director, Institute of Library Research; the project director and project manager were, respectively, Ralph M. Shoffner and Allan J. Humphrey, Institute of Library Research.

This report is being issued as seven separate volumes:

- Shoffner, Ralph M., Jay L. Cunningham, and Allan J. Humphrey. The Organization and Search of Bibliographic Records in On-line Computer Systems: Project Summary.
- Shoffner, Ralph M. and Jay L. Cunningham, eds. The Organization and Search of Bibliographic Records: Component Studies.
- Aiyer, Arjun K. The CIMARON SYSTEM: Modular Programs for the Organization and Search of Large Files.
- Silver, Steven S. INTX: Interactive Assembler Language Interpreter Users' Manual.
- Silver, Steven S. FMS: Users' Guide to the Format Manipulation System for Natural Language Documents.
- Silver, Steven S. and Joseph C. Meredith. DISCUS Interactive System Users' Manual.
- Smith, Steven F. and William Harrelson. TMS: A Terminal Monitor System for Information Processing.

Because of the joint support provided by the Information Processing Laboratory Project (OEG-1-7-071085-4286) for the development of DISCUS and of TMS, the volumes concerned with these programs are included as part of the final report for both projects. Also, the CIMARON system (which was fully supported by the File Organization Project) has been incorporated into the Laboratory operation and therefore in order to provide a balanced view of the total facility obtained, the volume is included as part of the Laboratory project report. (See Maron, M.E. and Don Sherman, et al. An Information Processing Laboratory for Education and Research in Library Science: Phase 2. ILR 1971.)

I N T X

Interactive Assembler Language Interpreter

Under URSA

Preliminary Programming Manual

Supported By

The Campus Computing Network
The Office of Education

Steven S. Silver
Institute of Library Research
University of California, Los Angeles, Berkeley

INTRODUCTION

INTX is an interactive programming and debugging system operating under UCLA's URSA interactive console system. Although originally designed as a debugging aid for interactive processor development, the addition of an on-line Assembler makes it a programming system in its own right.

HARDWARE

INTX operates only on the Computer Communications 301 graphics display device making use of automatic update and cursor positioning facilities.

ORGANIZATION

There are three major divisions in the system:

1) Command Analysis and Initialization,
2) Assembler-Loader,
3) Interpreter-Disassembler.

The rest of this document will describe the functions and attributes of each of these divisions. See figure 1 for a system organization chart.

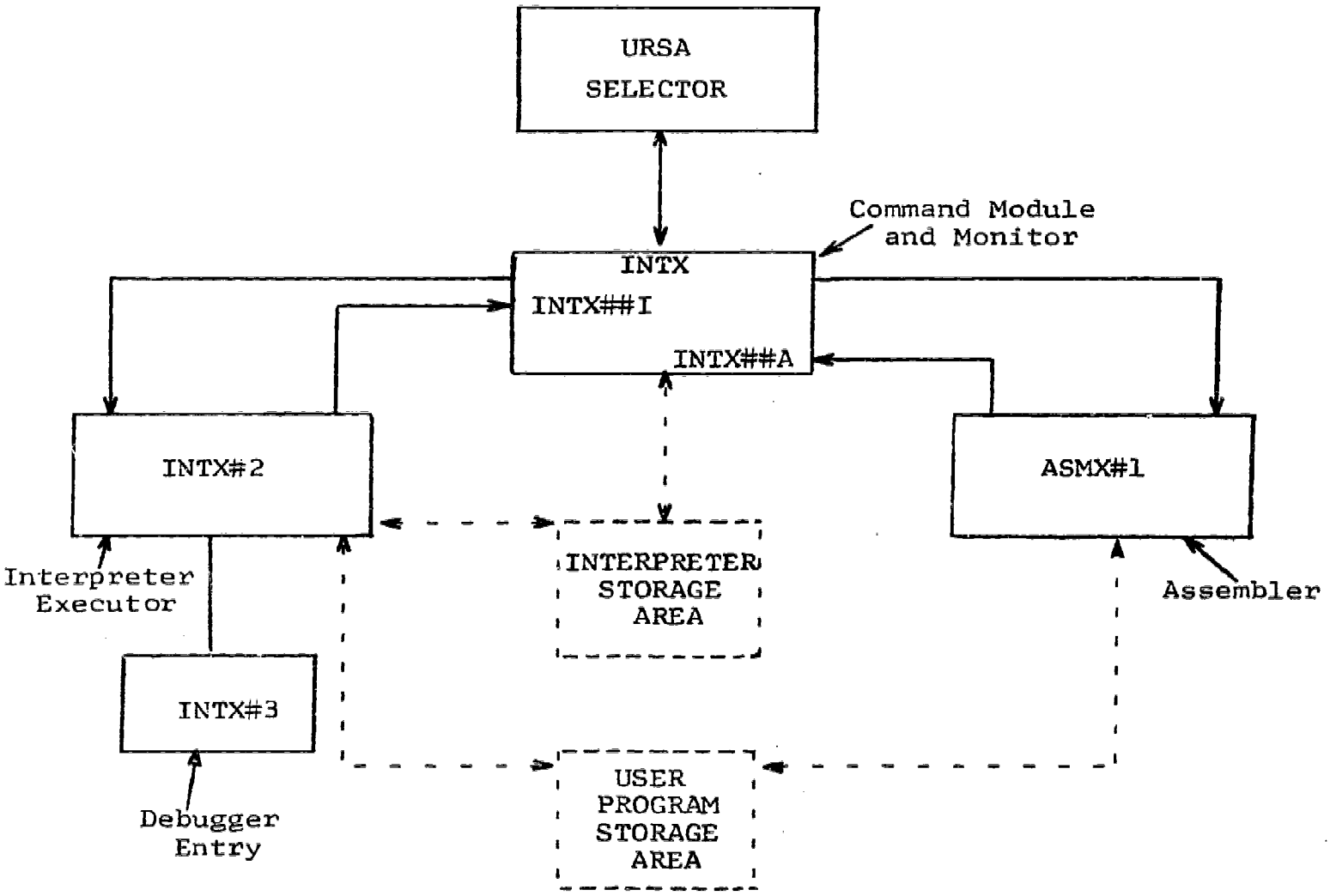
COMMAND MODULE (INTX)

The Command module displays an initial screen describing functions available and current level of development. The Command module also supervises the operation of all other components of the system.

When either the Assembler or Interpreter need control commands analysed they must transfer control to the command module. In the case of the Assembler, control is not passed back unless specificely requested. The Command module will pass control to the Interpreter after it has performed command analysis.

FIGURE 1

SYSTEM STRUCTURE



ASSEMBLER (ASMX#1)

The Assembler is a subset of Basic Assembler Language with extended branch mnemonics. The pseudo operations currently available are EQU, DS, and DC. The Assembler is invoked by typing "ASM" in the command module and it will not normally relinquish control until it has finished assembling. The source input data set must be a standard KEYPUNCH (URSA program editor) data set (i.e. DCB=(RECFM=FB,BLKSIZE=400,LRECL=80)).

ASSEMBLER EXTERNAL DESCRIPTION

The assembler accepts standard assembly language and the EQU pseudo operation. Any legal 360 Assembler statement will be accepted with the following restrictions:

1) No literals,
2) No implied lengths,
3) No multiplication or division in statement construction.

There is minimal error checking. Only undefined symbols will suspend assembler processing (until interrupt is pressed). The system has a maximum capacity of 100 symbols and a 2K over-all program length limitation.

INITIALIZATION

A program storage area into which the program will be loaded for execution is obtained.

The data set to be processed is identified by completing the partial data set name on the console screen. Pressing interrupt will begin processing.

PASS I

During the first scan of the input source a symbol table is generated and EQU statements are analysed. Register 15 is assumed to be the base register by this version of the Assembler.

PASS II

The operand field of each statement is analysed and symbolic references are resolved. Errors are ignored and a best guess is assembled in lieu of correct code. As each statement is analysed it is loaded into the program storage area.

When this pass is completed register 15 is set equal to the start of the program. Control is returned to the command module with a "start execution" command supplied.

SVCX

The INTX instruction set makes use of the SVCX operation to invoke certain system functions supplied by the Interpreter. It is a four byte instruction having a HEX 51 operation code. The low order bits describe the operation to be performed. The appendix gives a detailed explaination of the function of each SVCX thus far implemented.

The Interpreter module is designed to execute any System/360 instruction except the SVC. Any instruction that could be successfully executed by a program running in problem state is supported. If a program check occurs, a message describing the failure appears on the screen and processing is suspended. The program under interpretation does not control its own execution for more than one instruction.

SVC-SVCX

SVC's are not supported for the following reasons:

1) Control over a program is usually lost after an SVC is issued,

2) Many SVC's are highly installation and operating system release dependent, making implementation difficult,

3) System integrity can not be guaranteed during an SVC.

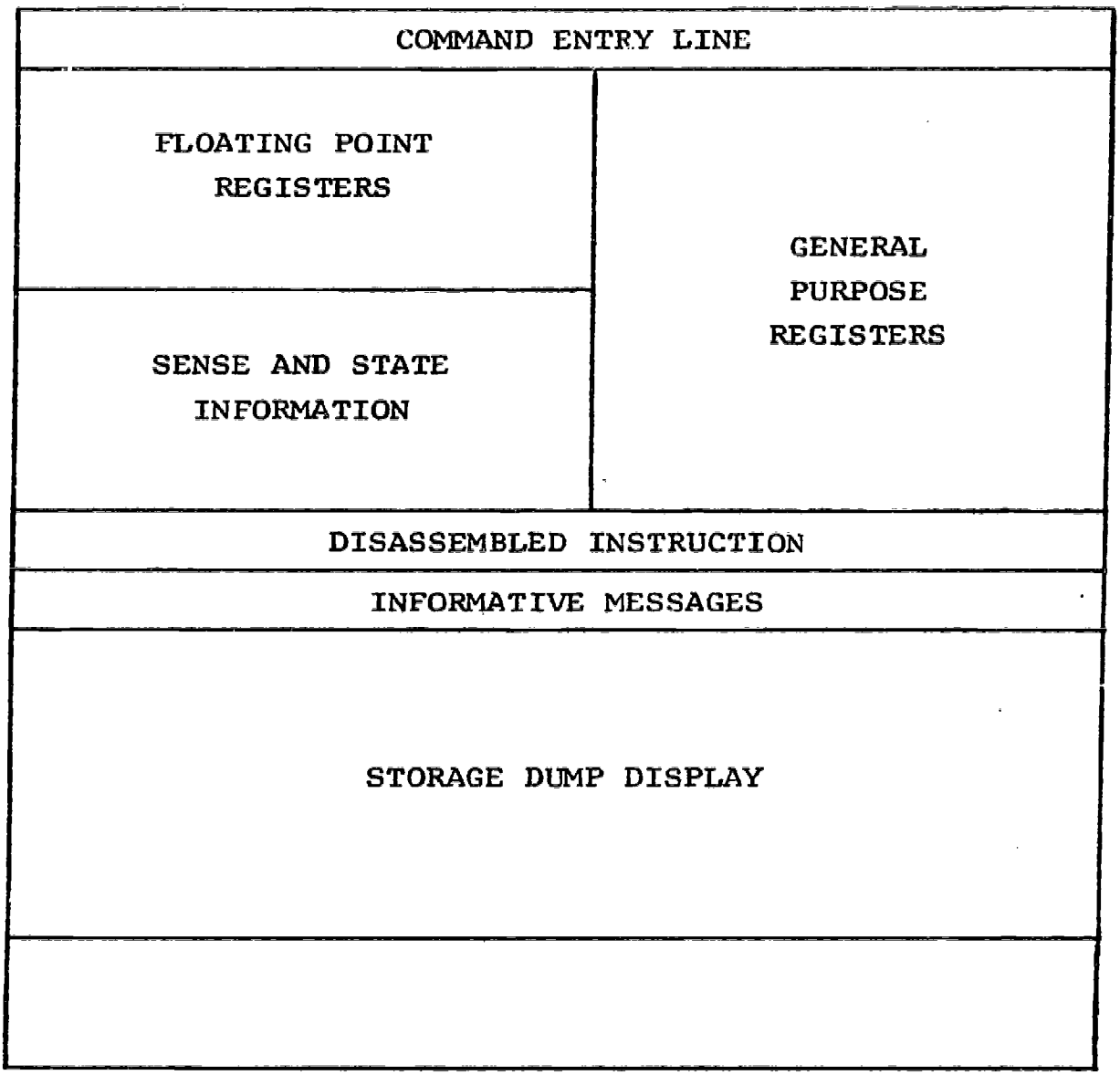
To provide functions usually performed by SVC's the SVCX has been defined. Detailed functional descriptions of this instruction will be found in the appendix. Since SVCX is a built-in function of the Interpreter its functioning code is not a part of the users program code.

COMMAND ENTRY

Control states within the Interpreter are usually modified by explicit command. Commands are entered on the top line of the screen. When interrupt is pressed the Command module will change the state of the Interpreter.

FIGURE 2

INTERPRETIVE DISPLAY LAYOUT



MACHINE DISPLAY

The Interpreter will display the 16 general purpose and 4 floating point registers, the condition code byte and instruction counter, the HEX and disassembled version of the instruction just executed, and a core dump of a selected area of storage. See figure 2 for the screen layout.

The progress of the execution of a program can be monitored during each execution cycle by entering YD (Yes Display) mode. This mode is automatically assumed if a program check occurs. High speed execution can be resumed if ND (No Display) is specified.

CORE DISPLAY CONTROL

The core dump display is controlled by commands which set the Dump's starting address directly, indirectly, based on the contents of a register, or based on the address of the instruction currently being executed. See the appendix for information on how to use these functions. The starting address of the display is always aligned to the next lowest full word boundary.

SPEED OF EXECUTION

The speed of display mode execution is controlled by the "T" (Time) command. Execution is delayed a specific number of seconds between instruction cycles. The next instruction can be executed by pressing interrupt or waiting for the time specification to elapse. Setting a large time estimate will essentially allow single stepping through a program. Turning off display mode will override the time specification and speed up execution.

EXECUTION CONTROL

The ranges of execution may be redefined but the area in which a program may store may not. Commands are available to modify the instruction counter and the contents of the general purpose registers.

STORAGE AND FETCH PROTECT

Programs are permitted to execute and store only within the boundaries defined by the physical code loaded by the Assembler. All instructions that store or fetch information from storage are validity checked to make certain they reference core only within the established bounds. Branch addresses are pre-calculated calculated and checked before use.

INTX uses the pecking order established in the URSA time sharing system in the following way; Non-systems programmers cannot access data (fetch) or branch outside of their program area. If systems programmers wish the fetch protection option they may use the PSET command.

INCONSISTENCIES

The Interpreter will give faithful results in all cases except the BAL and BALR instruction. It is possible in this case for the instruction length to be in error since not all instructions are directly executed. The condition code and branch address are correct.

OPERATION CODE ANALYSIS

A software defined instruction counter points to the instruction to be executed. The one byte operation code is used as an index into a table, each entry of which contains the character form of the operation, special flags controlling execution, and a displacement classifying each operation into one of ten types of instructions (in addition to error). This information is used by the Interpreter to disassemble the instruction.

DISASSEMBLY AND REGISTER ALLOCATION

Disassembly is based on the different formats for representing an instruction by the Assembler (i.e. RR, SI, SS, etc). As the instruction is being disassembled, registers are fed into an allocation scheme which converts the requested register into a real register reference. The resulting real register is not within GPR's 10 through 15. These 6 are reserved by the Interpreter for maintaining control over the program under execution.

By using an allocation technique the user is not required to use a subset of the real register set. All sixteen registers appear to be available for use.

In most cases the result of execution will be obtained with little actual Interpreter instruction modification. For example, any arithmetic operation not using a register above 9 will execute "as is".

EXECUTION

The parsed and reconstructed instruction is placed into an instruction storage block that is terminated by a BALR 15,10. Register 15 will contain, after execution, the resulting condition codes. Register 10 will be preloaded with the addresses to which control is to return to the Interpreter. If the instruction has not been flagged as being invalid the Interpreter will branch to the instruction block. Control is either returned to the address pointed to by register 10, passed to a branch address within the freshly reformed instruction, or passed to the SPIE routine after its execution caused a program check. Model 91 imprecise interrupts are correctly handled.

MICRO-PROGRAMMED INSTRUCTIONS

If the instruction is one of the twelve instructions that may either gain or seize control (BC, BXLE, etc) or which have strange properties (STM, EX, etc) then a pseudo micro-programmed version is executed in place of the instruction itself. The net result will be equivalent.

APPENDIX I COMMAND SUMMARY

IC=address.....Set the instruction counter to the value specified.

ICR=register.....Set the instruction counter to the value contained in the register specified.

(register)=value....Set a General Purpose Register.

T=value.....Set the speed of execution.

YD.....Allow the display of the state of the machine while interpreting.

ND.....Turn off the machine state display.

HE=address.....Set the high limit of execution.

LE=address.....Set the low limit of execution.

ASM.....Transfer control to the Assembler.

IX.....Interpret with execution.

INX.....Interpret and scan with no execution.

PSET.....Reset fetch protect for system programmers.

X.....Use experimental modules (RESTRICTED).

D=address.....Core dump display at address specified.

DI=address.....Core dump display starting at address pointed to by full word at address specified.

DR=register.....Core dump display starting at address contained in register specified.

DE.....Core dump display starting at the address pointed to by the instruction counter (default mode).

Address, register, and value information can be supplied by combining the operators + and -,*, and either decimal or

hexadecimal (base 16) data where decimal information must be followed by a period (.). * is the current value of the instruction counter when used to set a register or the instruction counter, or the last address displayed by the core dump routine. There must not be any embedded blanks in the command.

EXAMPLE: (1+C-12.)=*+FFF-15. is a valid input format for setting the value of register 1 equal to the current value of the instruction counter + FF0.

APPENDIX II INSTRUCTION CLASSES AND MICRO PROGRAMMED INSTRUCTIONS

CLASSES: (see IBM C28-6514-5 p117)

RR1	OP	R1,R2
RR2	OP	R1
RR3	OP	I
RX1	OP	R1,D2(X2,B2)
RS1	OP	R1,R3,D2(B2)
RS2	OP	R1,D2(B2)
SI1	OP	D1(B1),I2
SI2	OP	D1(B1)
SS1	OP	D1(L1,B1),D2(L2,B2)
SS2	OP	D1(L,B1),D2(B2)

.....

MICRO PROGRAMMED INSTRUCTIONS

SVCX BALR BAL BCR BC BCTR BCT BXH BXLE EX LM SPM STM

APPENDIX III SVCX INSTRUCTIONS

DIRECTORY:

SVCX.....FUNCTION

0.....Console Communication.

1.....Indirect execution of INTX commands.

2.....Non-fatal transfer to DEBUGGER (URSA SPIE routine).

SVCX 0 CONSOLE INPUT/OUTPUT

Calling Sequence:

```
        LA    1,LIST
        SVCX      0

LIST    DC    F'write_addr'
        DC    F'write_length'
        DC    F'read_addr'
        DC    F'read_length'
        DC    H'0' length of read
        DC    H'0' reserved
```

write_addr.....location of area from which data will be written starting at the top left-most corner of a screen after an erase.

write_length.....true length of write to be performed. If this is zero the write will be ignored.

read_addr.....location of area to which data will be transfered. Transfer will start from the location of the cursor as left by the previous write.

read_length.....the maximum number of characters to be transfered by the read. If zero no read will be done.

length read.....the number of characters actually read by an issued read.

reserved..... will contain status flags relative to console I/O operations.

NOTE:

All standard INTX storage and fetch protection will be observed in I/O operations.

SVCX 1 PROGRAM CONTROLLED COMMAND ENTRY

Calling Sequence:

```
     LA    1,LIST
     SVCX        1


LIST      DC    C'intx_commands'
          DC    X'0'
```

intx_commands.....Any command sequence that could normally be entered on the top (command) line of the INIX interpreter.

SVCX 2 CONTROLLED ENTRY TO DEBUGGER

Calling Sequence:

```
SVCX    2
```

COMMENT

INTX registers 2 through 10 are moved into the real registers and module iNTX#3 is entered which gets a 0C3 program check. INTX#2, the Interpreter, is not affected by the DEBUGGER entry.

APPENDIX IV ERROR MESSAGES

0C1	Operation	P
0C2	Priviliged-operation	P
0C3	Execute	P
0C4	Protection	P
0C5	Addressing	P
0C6	Specification	P
0C7	Data	P
0C8	Fixed-point-overflow	P
0C9	Fixed-point-divide	P
0CA	Decimal-overflow	P
0CB	Decimal-divide	P
0CC	Exponent-overflow	P
0CD	Exponent-underflow	P
0CE	Significance	P
0CF	Floating-point-divide	P
	Out of execution area	X
	Software detected protectiong	X
	SVC suppressed	X
	Bad SVCX instruction	X
	Object of execute	X
***** message waiting. Please type 'end'		
	Software fetch violation	
	Protection	I
	Addressing	I
	Specification	I
	Data	I
	Fixed-point-overflow	I
	Fixed-point-divide	I
	Exponent-overflow	I
	Exponent-underflow	I
	Significance	I
	Floating-point-divide	I

I N T X

Interactive Assembler Language Interpreter

Under URSA

Version II extensions

Supported By

The Campus Computing Network
The Office of Education

OEG-1-7-071083 Hayes

DISCUS Project

Steven S. Silver
Institute of Library Research
University of California, Los Angeles, Berkeley

SVCX 3 CARD IMAGE DISK INPUT/OUTPUT

Calling sequence:

```
          LA    1,LIST
          SVCX 3
```

```
LIST DC    A(address_of_80_byte_buffer)
     DC    F'relative_card_number'
     DC    A(card_not_found_exit)
     DC    A(addr_of_VOL_DSN_field)
     DC    X'type_of_I/O'
     DC    XL3 reserved
```

address_of_80_byte_buffer....DS 80C an area which holds the card being operated opon.

relative_card_number.....an integer which indicates the card to be operated opon (1 is the first card)

card_not_found_exit.....the address specified receives control of the card sequence number specified does not exit.

addr_of_VOL_DSN_field.....a 50 byte field having the following format:
DC CL6'volume_name' or XL6'00' for cataloged.
DC CL44'fully_qualified_data_set_name'

type_of_I/O.....a one byte field with one or more of the following bits on:

- X'01' read a card
- X'02' write a card
- X'04' reestablish the data set (i.e. reopen it based on the VOL_DSN field)

reserved.....reserved for future expansion

NOTE:

1) only one DSN-VOL field may be in effect at one time.

2) the first reference to a data set will be equivelent to having X'04' type specified in addition to the read or write flag.

3) storage and fetch protection are observed.

4) major errors cause "bad svcx instruction" messages to appear.

5) pre_established URSA KEYPUNCH data sets are used:
DCB=(RECFB=FB,BLKSIZE=400,LRECL=80)

6) blocks can not be added to the end of a data set by this SVCX.

SVCX 4 URSA CONSOLE SYSTEM DATA

Calling sequence:

```
          LA    1,LIST
          SVCX  4
```

```
LIST DS   F address_of_your_FCA
     DS   6C your_job_number
     DS   3C your_initials
```

address_of_your_FCA.....A full word pointing to the start of the 4K block of storage reserved for the console being used.

your_job_number.....The job number being charged for your console use.

your_initials.....The three byte form of the initials used to sign on to the console system. The last position will be blank if two character initials were used.

NOTE: Storage protection is observed.

SVCX 11 TIME OF DAY

Calling sequence:

```
SVCX 11
```

NOTE:

1) Equivelent to an OS macro: TIME DEC.

2) Registers 0 and 1 are modified by this SVCX.

Calling sequence:

```
        LA   1,LIST
        SVCX 35
```

```
LIST DC   H'length_of_message_+_4'
     DS   H
     DC   C'message'
```

length_of_message_+_4.....A half word containing the length of the complete data list.

message.....The message you wish to output to the operator console.

NOTE:

1) The message is typed on the operators console with the prefix message code "CSM300I".

2) Reserved exclusively for systems programmers.

SVCX-0- CHANGES-

1) An automatic time sequencing feature has been added:

```
DC   F'write_length'
```

is replaced by:

```
DC   H'time-out_wait'
DC   H'write_length'
```

Where "time-out_time" is the time in seconds to wait until a console interrupt is simulated. A real console interrupt overrides this parameter. A specification of less than three seconds is reset to 300 seconds (five minutes).

2) The full 800 character screen may now be used.

REGISTER NOTATION

The contents of the general purpose registers can be used in the calculation of values used by the command module. A single Hex or Decimal value surrounded by parenthasis will be treated as the contents of the equivelent register.

For example: IC = (14.)-(A)-3+* means set the instruction counter to the value contained in register 14 less the contents of register 10 less 3 plus the value of the instruction counter at the start of execution of the command.

STORAGE REFERENCE STOPS

To aid in program debugging three "stop on storage condition" commands are available. Each must be set by command to a particular absolute address location.

SI stops the machine when the instruction counter exactly matches the value specified

SS stops when a store at a particular location is completed.

SF stops when a fetch from a specified location is completed.

Each stop may be disabled by setting it equal to zero. The stop time is 1 minute or the last value of T, which ever is longer, as with all severe errors.

INSTRUCTIONS EXECUTED COUNTER

NX holds the number of successfully executed instructions since the last time NX was set. It can be set to any value by command and is initially set to zero when execution begins.

HIGH AND LOW VALUES OF STORAGE AND EXECUTION

All storage and fetch protection checking are based on the limits of storage which are now displayed. Only execution boundaries may be changed. Storage limits may not be changed.

DISPLAY OF SYMBOLIC INSTRUCTION COUNTER

If an INTX assembly was done to generate the code to be executed the symbol table from the assembly is left in core during the execution run. In addition to the Hex instrution counter the closest low value of labeled statement in the blank CSECT is also displayed with any appropriate displacements.

i.e.:

```
1     START      LA    R1,B(R2)

2                B     START
```

Statement 2 might generate the following information line:

```
                 BC    15,34(0,14)     START+4
```

FIRST LINE COMMAND ENTRY

During execution any command entered on the first line of the display screen will be executed no matter what the state of the program under execution. The cursor must be on the first line of the screen for data to be treated as command input.

MESSAGE HANDLING

When a message is sent to someone in INTX an explanitory error message will appear. The user may choose to ignore this if he wishes but it is advisable to leave INTX to find out the contents of the message.

FETCH PROTECTION EXTENSION

Anyone can now reference any area of low core -- lower than the address CVTNUCB in the CVT -- without a fetch protection exception.

INX and ND

INX will now suppress the execution of instructions and suppress the incrementing of the instruction counter (IC). Execution may continue by issuing either an IX or ND command.

ND now also implies the execution of IX. Execution

continues at high speed when ND is specified.

CHANGES TO THE ASSEMBLER

USING AND DROP

Usings are now required for all programs. Register 15 will still point to the start of the program but the user must supply a using indicating this fact. The using may specify up to 2 registers. Drops are limited to no more than 3 registers at a time.

START AND ORG

In this assembler ORG and START are exactly the same. Each modifies the instruction counter within its own csect (or dsect). As in the OS assembler a blank operand is assumed to refer to the highest value in the csect or dsect.

DSECT AND CSECT

A total of up to 10 csects and dsects may be used including the initial unspecified blank csect. Multiple csects are loaded in the order in which they are first referenced. Dsects are treated exactly like csects except loading is suppressed.

CONSTANTS

A, V and Y constants are now operational. Multiple constants may be specified in a single statement with the following restrictions:

1) commas may not be used within the quoted or parenthasised lists of arithmetic constants.

2) constants of the form nALm(*) will use the value *

calculated at the start of the expression and the same value will be propogated for each iteration.

ERROR MESSAGES

The assembler now issues error messages for most major syntactical errors along with a pointer to the approximate area of the error. Job control language in the intx program will not cause an error to be issued or executable code to be generated.

STORAGE REQUIREMENTS

Programs can vary in size between 2 and 8 thousand bytes. There is a fixed 4 thousand byte overhead for a symbol table passed to the executer for better debugging displays.

DATA SET CHECKING

Any data set with a second index level of CCN will be treated as a public data set (for assembly input only). Any user may reference a data set with this structure.